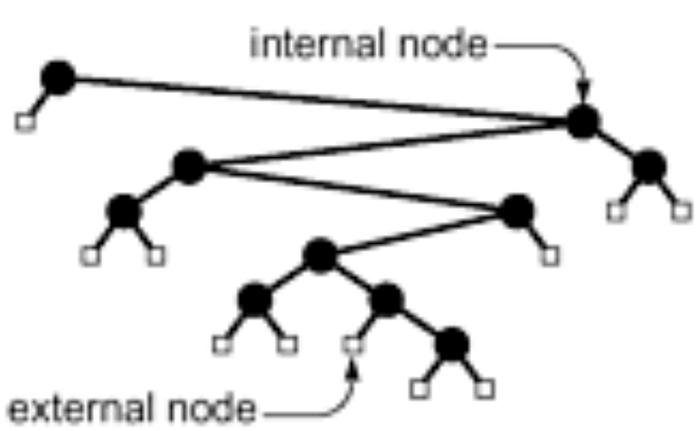
Arbres

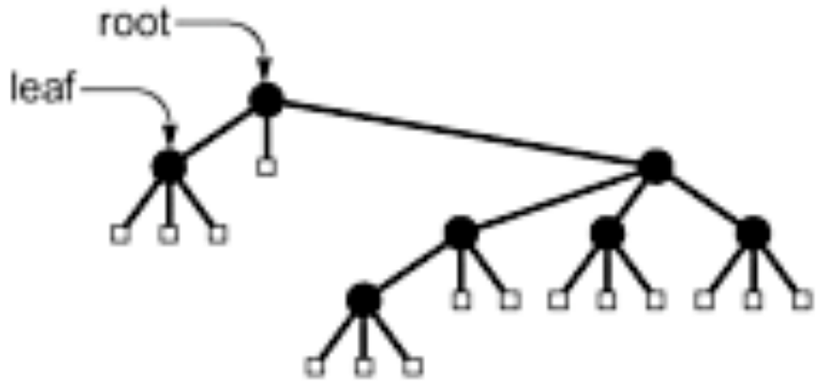
Réf. : Chapitre 7 du cours

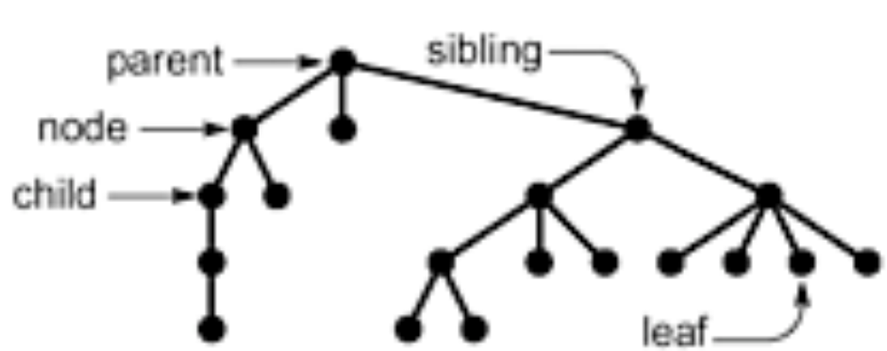
Pourquoi les arbres ?

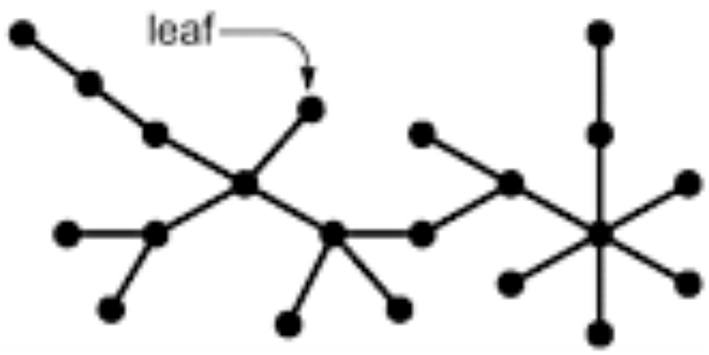
- Représenter des données déjà structurées en arbre (livre, famille, tournoi, langage)
- Parcourir des données de manière efficace (arbres binaires de recherche)

Types d'arbres



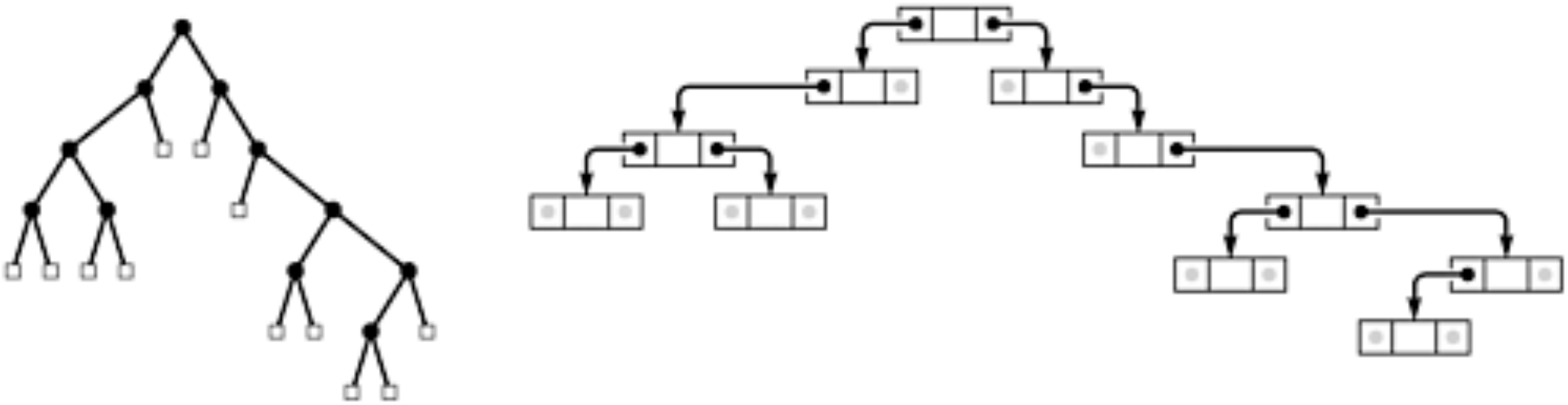






- binaire, ternaire
- arbre général/forêt (niveau = distance à la racine), arbre libre

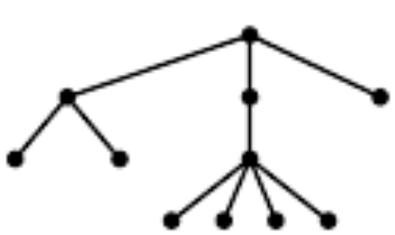
Arbre binaire: représentation

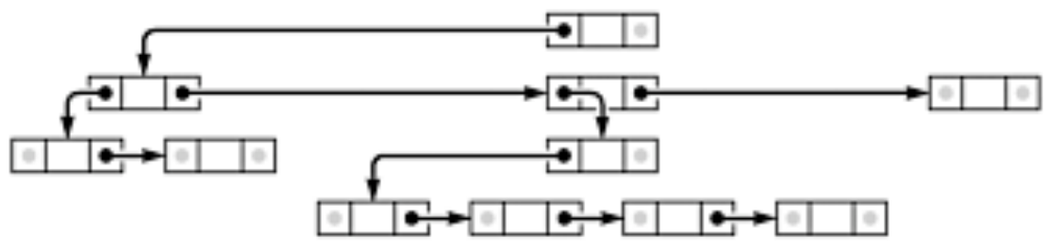


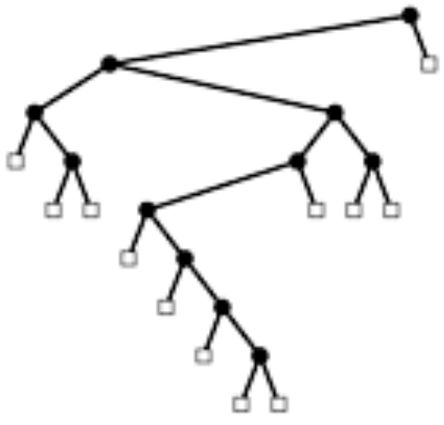
- Chaque noeud contient un lien vers les fils droit et gauche
- Les liens null correspondent aux noeuds externes

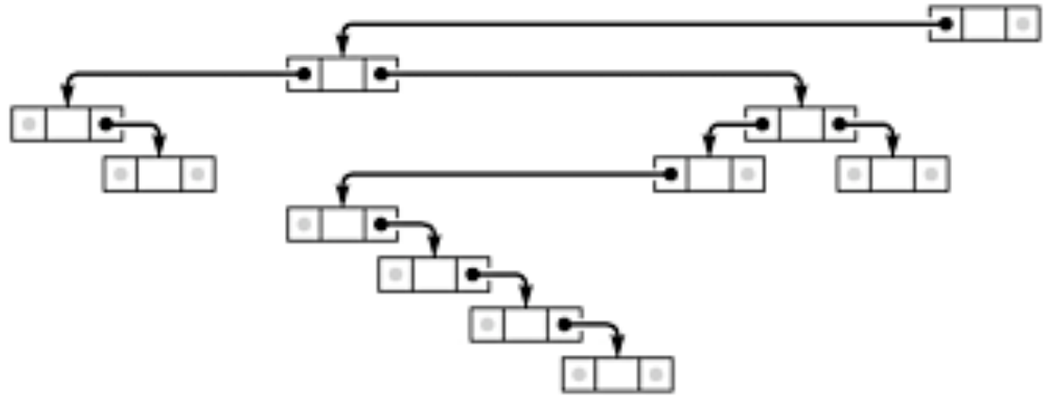
Arbres généraux : représentation

- Une liste chaînée de fils par noeud, ou bien...







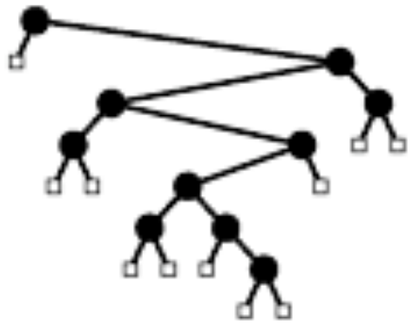


- Un arbre binaire !

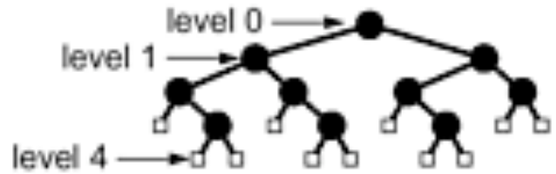
- Utilise la bijection fils-aîné_frère-cadet entre arbres généraux ordonnés de taille n+1 et arbres binaire de taille n

Définitions et propriétés

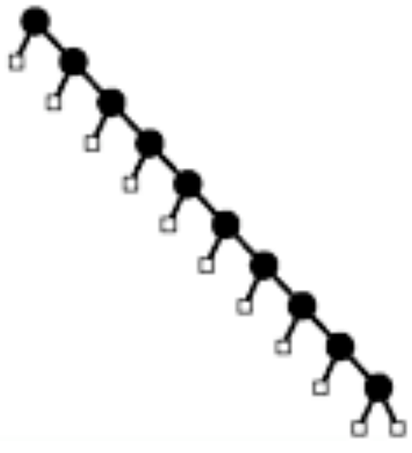
- **Niveau** d'un noeud = 1 +niveau de son parent (niveau racine = 0)



- **Hauteur** d'un arbre = niveau maximum des noeuds



- Hauteur d'un arbre binaire à N noeuds internes comprise entre log2N et N-1



Arbres : implantation

- enracine : info x arbre x arbre → arbre
- `type arbre is access noeud;`
- Arbre binaire :

```
type noeud is record
  info : type_info;
  fils_gauche, fils_droit : arbre;
end record;
```

- Arbre général :

```
type noeud is record
  info : type_info;
  fils_aine, frere_cadet : arbre;
end record;
```

Arbres : exploration en profondeur

- explore_et_traite (arbre_vide) =
  traitement_vide
- explore_et_traite (enracine (inf, ag, ad)) =
  combinaison (traitement(inf),
  explore_et_traite(ag),
  explore_et_traite(ad))

Arbres : exploration en largeur

- **Initialisation** :
  file := racine(arbre)

- **Boucle d'exploration** :
  tant que file /= file-vide repeter
  oter-en-tete (file, x);
  traiter (x);
  pour chaque fils y de x repeter
  inserer-en-queue (file, y);
  fin pour
  fin tant que

- **Complexité** en temps en O(N)

- **Taille mémoire** nécessaire = taille du dernier niveau ≤ (N+1)/2

- Possibilité de remplacer oter-en-tete et inserer-en-queur par des fonctions adaptées pour effectuer un parcours avec un ordre de priorité différent : selectionner-et-retirer et inserer-dans-ensemble

- **Exercice** : quelles fonctions "sélectionner" et "insérer" pour faire un parcours en profondeur ?

Retour sur les N reines

```
subtype pos_etendu is natural range 0..n ;
subtype pos is pos_etendu range 1..n ;
ligne : array(pos) of pos_etendu := (others => 0) ;

procedure Reine(i : pos_etendu) is
-- entree : i reines sont placées, mémorisées dans le tableau ligne
-- au retour, toutes les solutions complétant cette solution partielle
--   sont imprimées
begin
  if i=n then
    -- toutes les reines sont placées, on est à une feuille de l'arbre de recherche
    imprimer(ligne) ;
    return ;
  end if
  -- exploration des différentes possibilités
  k := i+1 ;
  for j in 1..n loop
    if position_libre (k,j) then
      ajouter_une_reine (k,j) ;
      reine(k) ;
      enlever_une_reine(k,j) ;
    end if ;
  end loop ;
end Reine ;

procedure ajouter_une_reine(k,j :pos) is
begin
  ligne(k) := j ;
end ajouter_une_reine ;

procedure retirer_une_reine(k,j :pos) is
begin
  ligne(k) := j ;
end retirer_une_reine ;

-- programme principal :
reine(0) ;
```

- Le parcours se fait-il en largeur ? en profondeur ?
- Comment implanter les heuristiques de recherche ?